System Programming Assignment 3

Due: 23:59 Tue, Dec 22, 2015

1 Problem Description

In assignment 3, you are required to deal with two scheduling problems between bidding_system and customers, one called **priority scheduling**, while the other called **earliest deadline first scheduling**. Customers will arrive at different time and bidding_system should deal with customers under these two different conditions.

There are two programs, **bidding_system** and **customer**. **Customer** will send customers to **bidding_system**, and **bidding_system** needs to notify **customer** after finishing any customer. We use some signals to indicate different customers.

1.1 Priority scheduling

There are three types of customer, ordinary customer, member customer and VIP customer. The priority order of customer is VIP > member > ordinary. That is, a customer with higher priority should be dealt with first when bidding_system receives him/her. If the priority of the processing customer is greater or equal to the arriving customer, then the arriving customer should be blocked until the processing customer finished.

Example 1: If an ordinary customer c_2 arrives at the time the bidding_system deals with an ordinary customer c_1, the bidding_system should finish c_1 first.

Example 2: if a VIP customer c_2 arrives at the time the bidding_system deals with an member customer c_1, the bidding_system should stop to deal with the VIP customer c_2.

Example 3: if an member customer or an ordinary customer c_2 arrives at the time the bidding_system deals with a VIP customer c_1, the bidding_system should finish a VIP customer c_1 first.

Table 1.1: customer information

customer code	type	process time(sec)	limit time(sec)
0	ordinary	1.0	no limit
1	member	0.5	1.0
2	VIP	0.2	0.3

[∗] You can use for loop and nanosleep() to simulate the time consuming, but you should notice that nanosleep() may be interrupted by signal.
[∗] If **customer** sends a VIP customer at 8:01:53.1, the customer should be sent back before 8:01:53.4

1.2 Earliest deadline first scheduling

There are three types of customer, ordinary customer, patient customer and impatient customer. In this case, whenever bidding_system receives any customer, it should calculate the customer's deadline with the following formula,

$$deadline = arrive\,time + limit\,time$$

After calculating the deadline of each customer, the one with earliest deadline should be dealt with first. If the deadline of the processing customer is earlier or equal to the deadline of the arriving customer, then the arriving customer should be blocked until the processing customer finished.

Table 1.2: customer information

customer code	type	process time(sec)	limit time(sec)
0	ordinary	0.5	2.0
1	patient	1.0	3.0
2	impatient	0.2	0.3

[∗] You can use for loop and nanosleep() to simulate the time consuming, but you should notice that nanosleep() may be interrupted by signal.
[∗] If **customer** sends a impatient customer at 8:01:53.1, the customer should be sent back before 8:01:53.4

2 Priority scheduling

2.1 Structure of program

There are two types of programs: **bidding_system**, **customer**

2.1.1 bidding_system

./bidding_system [test_data]

In the program, you should open a file named "bidding_system_log" which will be specified later. It is used to write some information during the execution of the program.

[test_data] is a file name and the content of the file is some information of testing data. It is also the argument of the **customer** program.

When **bidding_system** receives and starts dealing with the customer, it needs to write

receive [*customer code*] [*serial number*]

to file "bidding_system_log".

When **bidding_system** finished him/her, it needs to write

finish [*customer code*] [*serial number*]

to the file.

When **bidding_system** finishes the customer, it should also send a specific signal to **customer** and continue to deal with the next customer.(There may be some customer blocked. If no customer blocked, the **bidding_system** needs to wait until next customer arriving)

Bidding_system should fork one child and execute **customer** program. **Bidding_system** should use pipe to receive ordinary customer and redirect **customer**'s stdin/stdout to pipe.

The **bidding_system** uses the following signal to indicate **customer** which customer is finished.

Table 2.1: signals send to customer

type	signal
ordinary	SIGINT
member	SIGUSR1
VIP	SIGUSR2

When **bidding_system** read "EOF" from pipe, it should write "terminate\n" to file "bidding_system_log" and terminate the program itself.

2.1.2 customer

./customer [test_data]

In the program, you should open a file named "customer_log" which will be specified later. It is used to write some information during the execution of the program.

You should read the [test_data] and send the customer at the specific time in the [test_data].

When **customer** sends the customer, it needs to write

send [*customer code*] [*serial number*]

to file "customer_log".

When **customer** gets the finished customer(a specific signal) sent back by **bidding_system**, it needs to write

finish [*customer code*] [*serial number*]

to the file.

Notice that a customer should be received within a specific time. If **bidding_system** don't send back the customer within the specific time, the **customer** should write

timeout [*customer code*] [*serial number*]

to the file "customer_log" and terminate the program itself.

[*] In our test data, we assure that if **customer** sent a member customer, then it would not send another member customer before receiving the previous member customer(for avoiding signal missing). And so is the case of VIP customers.

Table 2.2: send to bidding_system

type	how to send customer
ordinary	ordinary\n
member	SIGUSR1
VIP	SIGUSR2

2.2 format of input and output

2.2.1 test_data

[*customer code*] [*sending time*]

The test_data file contains several lines. Each line has two numbers, the first number is a customer code corresponding to each type of customer(Table 1.1), and the second number is the sending time of the customer. The time is in ascending order in the test_data.

2.2.2 bidding_system_log

[*action*] [*customer code*] [*serial number*]

Table 2.3: bidding_system action

action	definition
receive	When bidding_system receives a customer
finish	When bidding_system finishes a customer

The customer code is the number corresponding to each type of customer.(Please refer to Table 1.1)

Each type of customer has each own serial number. For example, if the VIP customer is the second-place finish among all VIP customers, then his/her serial number should be 2.

When **bidding_system** read "EOF" from pipe, it should write "terminate`\n`" to file "bidding_system_log" and terminate the program itself.

2.2.3 customer_log

[*action*] [*customer code*] [*serial number*]

Table 2.4: customer action

action	definition
send	When customer sends a customer
finish	When customer receives a finished customer
timeout	When customer isn't sent back within time limit

The customer code and the serial number are the same as we defined in bidding_system_log.

[∗] In our test data, we assure that all actions will not at the same time. That is, if a customer is finished at 8:01:53.1, no signal will be sent at 8:01:53.1.

2.3 Sample execution

test_data
0 0
1 0.8
2 1.2
2 2.1
1 2.2

bidding_system_log
receive 0 1

receive 1 1
receive 2 1
finish 2 1
finish 1 1
finish 0 1
receive 2 2
finish 2 2
receive 1 2
finish 1 2
terminate

customer_log
send 0 1
send 1 1
send 2 1
finish 2 1
finish 1 1
finish 0 1
send 2 2
send 1 2
finish 2 2
finish 1 2

3 Earliest deadline first scheduling

3.1 Structure of program

There are two types of programs: **bidding_system_EDF**, **customer_EDF**

3.1.1 bidding_system_EDF

./bidding_system_EDF [test_data]

In the program, you should open a file named "bidding_system_log" which will be specified later. It is used to write some information during the execution of the program.

[test_data] is a file name and the content of the file is some information of testing data. It is also the argument of the **customer_EDF** program.

When **bidding_system_EDF** receives and starts dealing with the customer, it needs to write

receive [*customer code*] [*serial number*]

to file "bidding_system_log".

When **bidding_system_EDF** finished him/her, it needs to write

finish [*customer code*] [*serial number*]

to the file.

When **bidding_system_EDF** finishes the customer, it should also send a specific signal to **customer_EDF** and continue to deal with the next customer.(There may be some customer blocked. If no customer blocked, the **bidding_system_EDF** needs to wait until next customer arriving)

Bidding_system_EDF should fork one child and execute **customer_EDF** program. **bidding_system_EDF** should build pipe and redirect **customer_EDF**'s stdin/stdoutto pipe. When **bidding_system_EDF** receives "terminate\n" via pipe, it should write "terminate\n" to file "bidding_system_log" and terminate the program itself.

The **bidding_system_EDF** uses the following signal to indicate **customer_EDF** which customer is finished.

Table 3.1: signals send to customer_EDF

type	signal
ordinary	SIGUSR1
patient	SIGUSR2
impatient	SIGUSR3

[∗] Please add #define SIGUSR3 SIGWINCH to your code.

3.1.2 customer_EDF

./customer_EDF [*test_data*]

In the program, you should open a file named "customer_log" which will be specified later. It is used to write some information during the execution of the program.

You should read the [test_data] and send the customer at the specific time in the [test_data].

When **customer_EDF** sends the customer, it needs to write

send [*customer type*] [*serial number*]

to file "customer_log".

When **customer_EDF** gets the finished customer(a specific signal) sent back by **bidding_system_EDF**, it needs to write

finish [*customer type*] [*serial number*]

to the file.

Notice that a customer should be received within a specific time. If the customer didn't be sent within the specific time, the **customer_EDF** should write

timeout [*customer type*] [*serial number*]

to the file "customer_log" but do not terminate the program.

After reading "EOF" from [test_data], **customer** should send "terminate\n" to **bidding_system** via pipe.

[∗] In our test data, we assure that if **customer_EDF** sent a patient customer, then it would not send another patient customer before receiving the previous patient customer(for avoiding signal missing). And so is the case of ordinary and impatient customers.
[∗] In our test data, we assure that all actions will not at the same time. That is, if a customer is finished at 8:01:53.1, no signal will send at 8:01:53.1.

Table 3.2: send to bidding_system_EDF

type	signal
ordinary	SIGUSR1
patient	SIGUSR2
impatient	SIGUSR3

3.2 format of input and output

3.2.1 test_data

[customer code] *[sending time]*

The test_data file contains several lines. Each line has two numbers, the first number is a customer code corresponding to each type of customer(Table 1.2), and the second number is the sending time of the customer. The time is in ascending order in the test_data.

3.2.2 bidding_system_log

[action] *[customer code]* *[serial number]*

Table 3.3: bidding_system action

action	definition
receive	When bidding_system receives a customer
finish	When bidding_system finishes a customer

The customer code is the number corresponding to each type of customer.(Please refer to Table 1.2)

Each type of customer has each own serial number. For example, if the VIP customer is the second-place finish among all VIP customers, then his/her serial number

should be 2.

When **bidding_system** receives "terminate\n" via pipe, it should write "terminate\n" to file "bidding_system_log" and terminate the program itself.

3.2.3 customer_log

[*action*] [*customer code*] [*serial number*]

Table 3.4: customer action

action	definition
send	When customer sends a customer
finish	When customer receives a finished customer
timeout	When customer isn't sent back within time limit

The customer code and the serial number are the same as we defined in bidding_system_log.

[*] In our test data, we assure that all actions will not at the same time. That is, if a customer is finished at 8:01:53.1, no signal will be sent at 8:01:53.1.

3.3 Sample execution

test_data
0 0
1 0.4
2 0.8
2 2.1
1 2.2
0 2.6

bidding_system_log
receive 0 1
receive 1 1
finish 0 1
receive 2 1
finish 2 1
finish 1 1
receive 2 2
receive 1 2
finish 2 2
receive 0 2
finish 0 2

finish 1 2
terminate

customer_log
send 0 1
send 1 1
finish 0 1
send 2 1
finish 2 1
finish 1 1
send 2 2
send 1 2
finish 2 2
send 0 2
finish 0 2
finish 1 2

4 Grading

There are 6 subtasks in this assignment, you can get 8 points if you finish all of them.

4.1 Overall

1. (1pt)Produce executable files successfully.
   Your Makefile can generate *bidding_system* and *bidding_system_EDF* and *customer*.

4.2 Priority scheduling

1. (1pt)Your *bidding_system* can deal with ordinary customers.
2. (1pt)Your *bidding_system* can deal with ordinary customers and member customers.
3. (1pt)Your *bidding_system* can deal with all types of customers.
   (ordinary, member, VIP)
4. (2pt)You implement *customer* by yourself.
   You only need to implement *customer* based on priority scheduling (which would terminate if timeout).

4.3 Earliest deadline first scheduling

1. (1pt) Your *bidding_system_EDF* can deal with ordinary customers and patient customers.
2. (1pt) Your *bidding_system_EDF* can deal with all types of customers. (ordinary, patient, impatient)

5 Submission

Your assignment should be submitted to CEIBA before the deadline, or you will receive penalty.

At least 5 files should be included:

1. bidding_system.c
2. bidding_system_EDF.c
3. customer.c
4. Makefile(as well as other *.c files)
5. readme.txt

Since we will directly execute your Makefile, therefore you can modify the names of .c files, but Makefile should compile your source code(bidding_system.c, bidding_system_EDF.c, customer.c) into three executable files named *bidding_system*, *bidding_system_EDF* and *customer*.

In readme.txt, it should contain two parts.

1. Execution
   Please teach us how to run your program in case we could not run your code successfully.
2. Self-Examination
   Please write down which grading criteria do you finish. We will grade your assignment by the points you mentioned.

These files should be put inside a folder named with your student ID (in lower case) and you should compress the folder into a .tar.gz before submission. Please do not use .rar or any other file types.

The commands below will do the trick. Suppose your student ID is b03902000:

```
$ mkdir b03902000
$ cp Makefile readme.txt *.c b03902000/
```

```
$ tar -zcvf SP_HW3_b03902000.tar.gz b03902000/
$ rm -r b03902000/
```

Please do NOT add executable files to the compressed file. Errors in the submission file (such as files not in a directory named with your student ID (in lower case), executable files not named correctly, and so on) may cause deduction of your credits. Submit the compressed file SP_HW3_b03902000.tar.gz to CEIBA.

6 Reminder

1. Plagiarism is STRICTLY prohibited.
2. Your credits will be deducted 5% for each day delay, but a late submission is still better than absence.
3. If you have any question, you can ask questions at the board SysProgram on PTT2, contact us via email or come to R302.
4. Check whether your questions have been asked on PPT2 before you send an email or come to ask TAs.
5. Please start your work ASAP and do not leave it until the last day!